Guaranteed On-Demand Discovery of Node-Disjoint Paths in Ad Hoc Networks

Changwen Liu, W. Steven Conner, Mark D. Yarvis, and Xingang Guo
{changwen.liu, w.steven.conner, mark.d.yarvis, xingang.guo}@intel.com
Intel Corporation
2111 NE 25th Ave, Hillsboro, OR 97124

***Abstract*—In recent years, the low cost and abundance of WLAN products has led to the deployment of self-configuring multihop ad hoc networks. Multipath routing has been increasingly studied to improve network reliability and throughput. However, no existing work guarantees discovery of node-disjoint paths when they exist, which limits their applicability in real networks. This paper presents a theoretical framework that establishes the equivalence between multipath discovery and flow network assignment. This equivalence is used to guarantee the on-demand discovery of an arbitrary number of node-disjoint paths between a pair of nodes as long as they exist. We also present an example protocol that integrates the theoretical framework with the Dynamic Source Routing (DSR) protocol to find two node-disjoint paths, which can be easily extended to finding *k* node-disjoint paths. Analysis of the example protocol demonstrates a good tradeoff between complexity and capability, particularly when compared with existing on-demand multipath routing protocols.**

Keywords: Graph Theory.

I. INTRODUCTION

In recent years, the low cost and abundance of WLAN products has led to the deployment of self-configuring multihop ad hoc networks that provide wireless access across buildings, campuses, and large community areas [2] [12]. An example topology snapshot of MIT's RoofNet network [7] from June 12, 2004 is shown in Figure 1. In these networks, end-user devices participate as ad hoc network routers to forward traffic for neighboring nodes. Thus, wireless range and coverage can be extended each time a new wireless device is added to the network. However, as the number of devices increases in the network, wireless router failure (e.g. due to device malfunction) will be more frequent. In addition, because these wireless routers are controlled by individual users of the network, they are typically not as reliable as dedicated infrastructure. Users may choose to power-off, move, or reconfigure a node with no notice to other users of the network, disrupting data communication that is flowing along a route through the node.

On-demand routing protocols are well suited for wireless routing in large, random, and dynamic multihop ad hoc networks because they only maintain the routes that are in active use. Two of the most popular on-demand routing protocols for ad hoc networks are the Dynamic Source Routing (DSR) protocol [4] and the Ad hoc On-demand Distance Vector (AODV) protocol [10]. However, both are designed to find and maintain only a single path between a pair of nodes at any given time. Thus, when router failures occur, the latency of route repair—propagation of route error (RERR) information and establishment of a new route with route requests (RREQ) and route replies (RREP)—can disrupt communication for an extended period of time.

Multipath routing can improve the robustness of an ad hoc network by establishing multiple paths between source-destination pairs that can be used for rapid failover. Multipath routing can be classified into two categories: link-disjoint and node-disjoint. Both types of multipath routing provide improved reliability, load balancing, and the end-to-end throughput improvement. Pham *et al.* [11] show that using multipath routing in high-density ad hoc networks results in better throughput than using unipath routing. Node-disjoint routing

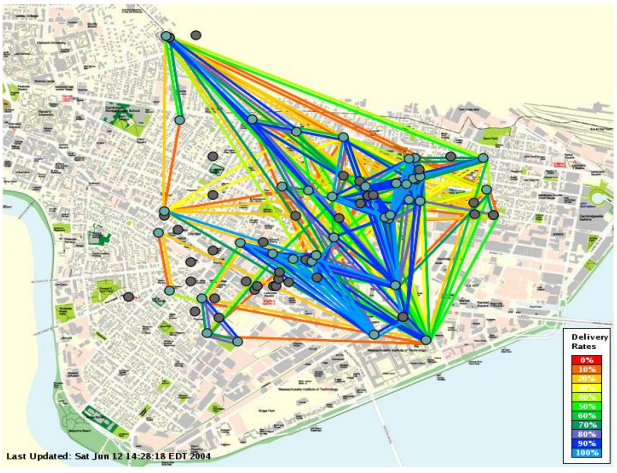

Figure 1: Connectivity map of the MIT RoofNet network.

* Other names and brands may be claimed as the property of others.

provides additional benefits of enhanced robustness to node failures and congestion, and enhanced capability for load balance. In this paper, we focus on node-disjoint paths, which are particularly beneficial in community-area networks. In these networks, nodes can be turned off, rebooted, or moved by their individual owners without notice. Thus, node failures can have a significant impact on the reliability of these networks.

A node-disjoint multipath routing protocol should identify two or more node-disjoint paths, if they exist, even in sparse network deployments. For example, consider the graph shown in Figure 2, which was derived from a snapshot of MIT's RoofNet network topology [7] on June 12, 2004 (a geographic topology snapshot from the same day is shown in Figure 1). All links for which reliability falls below 90% in either direction have been eliminated. Our goal is to ensure that two node-disjoint paths between nodes 5 and 23 can be identified. We will show in Section II that *existing algorithms fail to identify two distinct routes* in this real-world example. While this is just a single example, the ease with which we identified this example in a real network suggests that this situation is relatively common in community networks and perhaps other networks as well.

This paper first provides summary and analysis to prior work in node-disjoint multipath routing in Section II. We focus on the inability of existing protocols to guarantee identification of node-disjoint paths in networks like Figure 2. We then present a theoretical framework in Section III for node-disjoint multipath routing for ad hoc networks that guarantees the discovery of node-disjoint paths if they exist. An extension to the DSR protocol for finding two node-disjoint paths using this framework is provided as an example implementation in Section IV, and we demonstrate its ability to find disjoint paths in Figure 2 and other challenging cases. We describe how this example protocol can be easily extended to find k node-disjoint paths. Finally, we analyze the complexity and performance measurement of the proposed algorithm in Section V. We demonstrate that despite its added guarantee the proposed algorithm has lower overhead than most existing node-disjoint multipath algorithms.

II. Prior Work and Their Limitations

Multipath routing has received substantial attention in wired networks with tabled-based routing protocols [13] [16]. However, since this class of proactive routing protocols is not well-suited for the applications and deployment scenarios described in Section I, we focus instead on on-demand ad hoc networking routing protocols. There have been several efforts to extend on-demand ad hoc routing protocols for discovering multiple paths, including both node-disjoint [5] [14] [15] and link-disjoint [9] [8] [6] approaches. Each approach is based on either DSR or AODV, and each tries to discover multiple paths with one modified RREQ flooding. Because we focus on node-disjoint multipath routing in this paper, we present in detail only the protocols that are designed for discovering multiple node-disjoint paths. We demonstrate below that with the example topology of Figure 2 (described in Section I) none of the existing approaches can *guarantee* the discovery of node-disjoint paths when such paths exist in the network. In the example topology, it is possible to have at most two simultaneous node-disjoint paths between node pair 23 and 5. For example, the paths 23→3→37→7→1→5 and 23→21→11→0→5 are two node-disjoint paths between 23 and 5, but none of the following existing protocols are able to guarantee the discovery of these paths. This is also true when finding multiple node-disjoint paths between the node pairs {23,2}, {8,2}, and {8,5}.

Lee *et al.* [5] present one way to extend DSR for finding either node-disjoint or link-disjoint paths using a scheme named Split Multipath Routing (SMR). SMR relaxes the forwarding rule for RREQ messages as follows: RREQs that are received through a different incoming link, and whose hop count is not larger than the previously received RREQs are broadcasted again by the node. However, to establish maximally disjoint routes, only the destination node can generate RREPs, even if intermediate nodes know a route to the destination. At the destination, the first received RREQ represents the shortest-delay path. From the subsequently received RREQs, the path that is maximally disjoint from the shortest delay path is selected. Although not specified, SMR should be able to be extended to select-

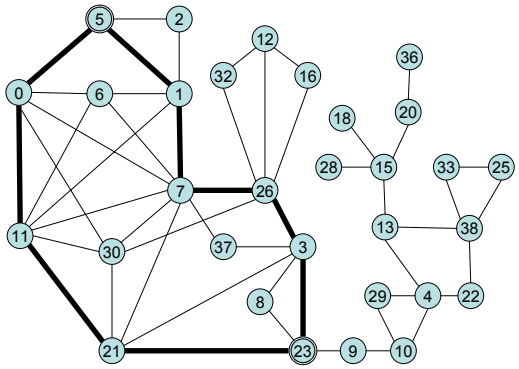

Figure 2: Topology graph from the RoofNet network with an instance of node-disjoint paths highlighted.

ing more than two routes. Taking Figure 2 as an example topology, when the RREQ traverses through 23→3→(37 or 26), it will have a higher hop count when it reaches nodes 7 and 30 than the RREQ that traverses through 23→21, and thus nodes 7 and 30 will drop the RREQ going through 23→3→(37 or 26). As a result, the destination 5 will only receive the RREQ going through 23→21 and hence no node-disjoint paths will be discovered between source 23 and destination 5.

Wu *et al.* [14] propose another way to extend DSR for discovering node-disjoint paths as follows: a RREQ is forwarded only if it is the first RREQ received for this round or the path included in this RREQ message is node-disjoint with the paths included in previously cached query messages for the same route request. However, in the example topology from Figure 2, this discovery of node-disjoint paths depends on the non-deterministic timing of RREQ message forwarding. In many situations it will fail to so identify existing node-disjoint paths. For example, consider the case in which node 0 receives its first RREQ from node 30, and node 1 receives its first RREQ from node 7. Without loss of generality, assume node 1 forwards the RREQ before node 0, therefore node 5 chooses path 23→21→7→1→5 as the shortest-delay path. As a result, the RREQ via the path 23→3 will be dropped by nodes 0, 1, and 5. Hence, in this situation no two node-disjoint paths will be discovered between node 23 and node 5. Figure 3 gives another ad hoc network example where the protocol in [14] will fail to discover two node-disjoint paths independent of the timing sequence of RREQ forwarding.

Ye *et al.* [15] describe AODVM, which extends AODV for discovering multiple node-disjoint paths as follows. First, intermediate nodes receiving duplicate RREQs record the information contained in these packets in a RREQ table. Unlike AODV, relay nodes are precluded from sending an RREP message directly to the source. Second, the destination replies to each received RREQ packet. Relay nodes forward a received RREP packet to the neighbor in their RREQ table that is along the shortest path to the source. To ensure that nodes do not participate in more than one route, whenever a node overhears one of its neighbors broadcasting an RREP packet, it deletes that neighbor from its RREQ table. In the example topology from Figure 2, for node-disjoint paths to exist between the source node 23 and the destination node 5, one path must include node 0, while the other path must include node 1. Similarly, near the source, one path must traverse through node 3, while the other path traverses through node 21. With AODVM, the shortest-delay paths from the source node

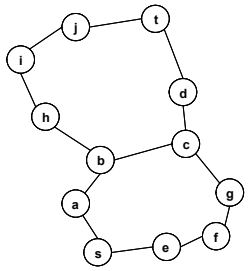

Figure 3: An example ad hoc network topology.

23 to both nodes 0 and 1 include link 23→21. Thus, a path that includes node 3 such as 23→3→37→7 will never be utilized by AODVM. Hence no two node-disjoint paths between nodes 23 and 5 can be discovered.

The failure of the three protocols on the two ad hoc networks is more than a coincidence. For the example topology, notice that there exist cuts {0, 1} and {7, 11, 30} between the source node 23 and the destination node 5. In any of the three protocols, RREQs traversing through alternative paths from source to the cut are usually dropped by nodes either in the cuts or succeeding the cuts. Similarly, RREPs will also be forwarded along the shortest path in the reverse direction. RREPs along alternative paths are usually dropped for the same reason. This leads to a critical insight into the designing of our theoretical framework: the first path discovered may preclude the discovery of other node-disjoint paths. To solve this problem, new paths may need to be merged with existing paths, and path reorganization may be necessary.

III. THEORETICAL FRAMEWORK

Before proposing enhancements to existing ad hoc routing protocols, we first establish a theoretical foundation for node-disjoint path discovery. As we will demonstrate in this section, the problem of finding node-disjoint paths in an ad hoc network is equivalent to the flow assignment problem in a flow network. In particular, finding k node-disjoint paths in an ad hoc network is equivalent to finding a flow assignment of value k in a corresponding unit-capacity flow network. Flow networks have been studied extensively in the past; many algorithms have been proposed which can deterministically tell whether a flow assignment of certain value exists [1]. Using a reverse mapping from the flow assignment back to node-disjoint routes, we can determine whether node-disjoint paths exist in the ad hoc network.

Of the many flow assignment algorithms, the iterative Ford-Fulkerson method possesses unique properties which not only allow the determination of such existence, but also facilitate the construction of such paths

on-demand, if they exist. In the Ford-Fulkerson method, each iteration attempts to increase the flow value in the unit-capacity flow graph by 1. If the flow value has not reached the maximum flow value, an *augmenting path*, which is a simple path from source to sink in the residual network of the flow, can be identified on-demand. A unit-value flow along the augmenting path is then defined and added to the original flow, which results in a new flow with flow value increased by one. Once the maximum flow value is reached, the iteration will terminate and signal that no *augmenting path* can be identified.

With the aid of a reverse mapping from the flow network to the ad hoc network, the augmenting path and the associated flow along the path can be mapped to an auxiliary path in the original ad hoc network. However, the auxiliary path isn't automatically node-disjointing with paths already discovered. Hence, we also derive a merging procedure that reorganizes the auxiliary path along with the existing paths to generate a node-disjoint group of paths.

Our theoretical framework for node-disjoint path discovery consists of three steps. First, we transform the connectivity graph of an ad hoc network into a corresponding unit-capacity flow network. Second, we map the set of existing node-disjoint paths in the ad hoc network to a flow in the flow network. We apply the Ford-Fulkerson iteration on the flow network and the flow to identify at least one augmenting path and a unit-value flow along that path. Third, we construct an auxiliary path in the ad hoc network for each augmenting path and the unit-value flow along it, and we merge the auxiliary path with existing paths to derive a new set of node-disjoint paths with the number increased by one in the original ad hoc network. Steps 2 and 3 above can be repeated until no more augmenting paths are discovered.

The rest of the section is organized as follows: in III.A we present the mechanism that transforms an ad hoc network into a flow network. In III.B, we prove that node-disjoint path discovery is equivalent to the flow assignment problem. In III.C we apply the Ford-Fulkerson method and analyze the properties of the augmenting path generated by the method. Finally in III.D, we present an algorithm that constructs an auxiliary path on-demand in the ad hoc network based on the augmenting path. We also describe a procedure for merging an auxiliary path with existing node-disjoint paths such that the new set of paths are guaranteed to be node-disjoint and the number of resulting node-disjoint paths is increased by one .

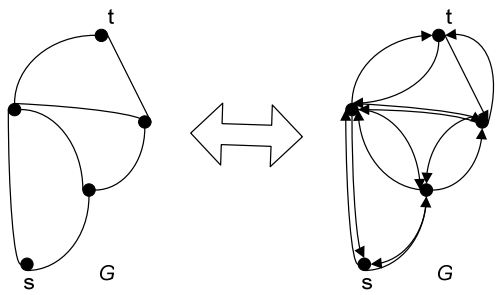

Figure 4 Equivalent graphic representations of an ad hoc network.

A. Ad Hoc Network to Flow Network

The connectivity of an ad hoc network can be represented by graph $G=(V,E)$, as demonstrated in Figure 4. While V is the set of nodes (or devices) in the network, an edge between node u and v in an undirectional graph indicates a bidirectional communication link between the two nodes with desirable quality, i.e. usable link, or a directional communication link from u to v in a directed graph. As it is self-explanatory, by default, we utilize a directional graph representation for an ad hoc network unless explicitly stated. A path from source s to destination t in the network can be represented by a path from s to t in the connectivity graph. The *destination* node is also referred to as the *sink* node. We will use the two terms interchangeably in this paper.

A flow network [1] is a directed graph $G^F=(V^F,E^F)$ with a source node s and a sink node t. Each edge in E^F has a capacity, indicated by a nonnegative value, and a flow value. The flow value can never exceed its capacity, although it can be zero or negative. When every edge has a capacity value of 1, the graph is called unit-capacity flow network. A flow assignment is the assignment of flow values to each edge that satisfies the capacity constraint, skew symmetry, and flow conservation as defined in [1]. Intuitively, a flow value of x on edge (u,v) indicates a net flow of x units from node u to v. A flow assignment corresponds to a set of edges with non-zero flow values from source s to destination t, and the flow value of this assignment is the maximum net flow that can be *pushed* from s to t along these paths.

The first piece of the theoretical framework is a procedure to transform a connectivity graph $G=(V,E)$ of an ad hoc network to its corresponding unit-capacity flow network $G^F=(V^F,E^F)$. Each node in G is split into two nodes in G^F, representing the ingress node and the egress node, respectively. Each directed edge in G is represented by a corresponding directed edge in G^F from an egress node to an ingress node. The detailed procedure follows:

1. Let $s,t \in V^F$. For each $v \in V - \{s,t\}$, $v^{(in)}, v^{(eg)} \in V^F$, referring to the ingress and egress nodes, respectively. $v^{(in)}$ and $v^{(eg)}$ are called *companion nodes* of v, and v is called the *root node* of $v^{(in)}$ and $v^{(eg)}$.
2. For each $v \in V - \{s,t\}$, $(v^{(in)}, v^{(eg)}) \in E^F$. Therefore, each ingress node is connected to its companion egress node by a directional edge.
3. For each edge $(u, v) \in E$, where $u, v \notin \{s, t\}$, $(u^{(eg)}, v^{(in)}) \in E^F$. Thus, each directed edge in E is represented by a directed edge from an egress node to an ingress node in E^F.
4. For each $(s, u) \in E$, $(s, u^{(in)}) \in E^F$. Similarly, for each $(u, t) \in E$, $(u^{(eg)}, t) \in E^F$. This rule covers the cases of s and t that are not covered in rule 3.
5. For each $(p,q) \in E^F$, $c(p,q) = 1$. Each edge in G^F has unit capacity.

Figure 5 shows an example of transforming G to G^F. Notice that there is a 1:1 mapping relation between a path from s to t in G and a similar path in G^F. However, the path in G^F has additional links between the ingress and egress nodes of each root node.

B. Node-Disjoint Paths and Flow Assignment

Once the connectivity graph G of an ad hoc network is transformed into a flow network G^F, we demonstrate that k node-disjoint paths exist in G if and only if a flow assignment of value k exists in G^F. We present the following two lemmas to prove the sufficient and necessary conditions, respectively.

Lemma 1 *Assume $T=\{P_i=(u_{i,,0}=s, u_{i,,1}, \ldots, u_{i,,n}, u_{i,n+1}=t), i = 1, \ldots, k\}$ are a set of k node-disjoint paths between source s and sink t in G. Define an integer-valued flow function $f: V^F \times V^F \rightarrow I$ on the flow network G^F as*

1. *For all j in $[1, n_i]$, $f(u_{i,j}^{(in)}, u_{i,j}^{(eg)}) = 1$, and $f(u_{i,j}^{(eg)}, u_{i,j}^{(in)}) = -1$.*
2. *For all j in $[0, n_i]$, $f(u_{i,j}^{(eg)}, u_{i,j+1}^{(in)}) = 1$, and $f(u_{i,j+1}^{(in)}, u_{i,j}^{(eg)}) = -1$.*
3. *For any other edge e in E^F, $f(e) = 0$.*

*Then f is a flow with value k on G^F. We refer to the flow f as the derived flow of the k node-disjoint path*s.

Remark: Essentially the derived flow of the k node-disjoint paths is defined by pushing a unit flow along each of the node-disjoint paths. Figure 6 gives a simple illustration for the lemma.

Proof: By definition of the flow network G^F, there exist k node-disjoint paths in the G^F, directly mapped from the k node-disjoint paths of the ad hoc network graph G. The flow f defines a net flow along the k node-disjoint paths in G^F. Obviously flow f satisfies the capacity constraint, skew symmetry, and flow conservation with the value $|f| = k$. □

The reverse condition of Lemma 1 is as follows.

Lemma 2 *For an integral flow f with flow value k on G^F, there exist k node-disjoint paths in G between source s and sink t.*

Intuitively, a flow value k indicates the existence of paths in G^F consisting of edges with flow assignment of 1 from s to t. Since G^F is a unit-capacity graph, each path can provide a flow value of 1. Hence, there must exist several such paths. In such paths, if one of a companion node pair is on the flow path, the other one must also be on the same flow path as the predecessor or the successor by the definition of the companion nodes. Assume there are two paths joined at a particular intermediate node. The paths will either share the previous link (joining at an egress node) or the next link (joining at an ingress node). This situation contradicts the flow conservation property, which specifies that the inflow and outflow of each node (except s and t) must be equal. So these paths must be node-disjoint. Since each path contributes a flow value of 1, there must exist k node-disjoint paths in G^F, which corresponds to k node-disjoint paths in G. The detailed proof is omitted due to space constraints.

From Lemmas 1 and 2, we can conclude that the existence of k node-disjoint paths between source s and sink t in G is equivalent to finding a flow f with flow value k

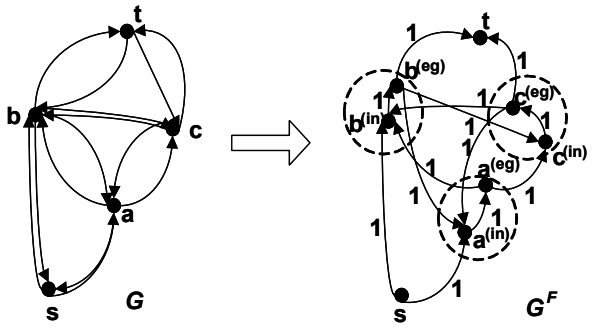

Figure 5 Derive a flow network from an ad hoc network G.

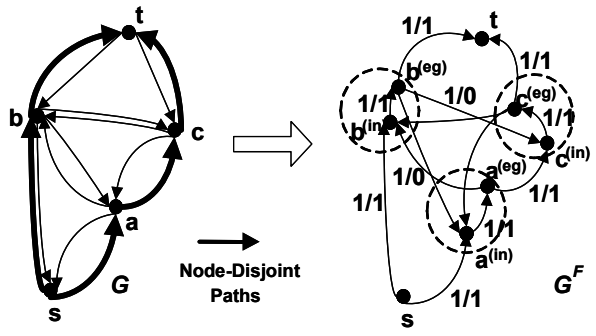

Figure 6 Derive a flow with value 2 in G^F from two node-disjoint paths in G

in G^F.

C. Ford-Fulkerson Method and Augmenting Paths

Now let us focus on the flow network. The Ford-Fulkerson method [1] is an iterative method for finding maximal flow between the source and sink in a flow network. It repeatedly augments the flow along augmenting paths until a maximum flow has been found. The algorithm has an important property: each time an augmenting path is identified, the flow value is increased. Moreover, with a unit-capacity graph and integral flows, each augmenting path will increase the flow value by exactly 1.

In this section, we will demonstrate that the augmenting path discovered by each Ford-Fulkerson iteration in G^F and that the flow along the path can be mapped to a path in the original ad hoc network G. To see what an augmenting path in G^F and the flow along it are mapped to in G, we first discuss some properties of an augmenting path in G^F. Let $Res_f(G^F)$ denote the residual graph of the flow f. By the definition of G^F, for each edge $e_i = (u, v) \in Res_f(G^F)$, there are two cases:

1. $f(u, v) = 0$ ⇨ If u and v have the same root vertex, then u is the ingress vertex and v is the egress vertex. If u and v have different root vertex, then u is the egress vertex and v is the ingress vertex.
2. $f(v, u) = 1$ ⇨ If u and v have the same root vertex, then v is the ingress vertex and u is the egress vertex. If v and u have different root vertex, then v is the egress vertex and u is the ingress vertex.

Thus, an augmenting path in G^F can be precisely characterized by the following lemma.

Lemma 3 *Let f be an integral flow in G^F, and $P = (u_0=s, u_1, u_2, \ldots, u_n, u_{n+1}=t)$ be an augmenting path in the residual network $Res_f(G^F)$ for the flow f. Then for each edge $e_i = (u_{i-1}, u_i)$ with $i \in [1, n+1]$, one of the following two cases must apply:*

1. *f has no flow between u_{i-1} and u_i in G^F, i.e. $f(u_{i-1}, u_i) = f(u_i, u_{i-1}) = 0$ in G^F.*
2. *f has unit positive flow in the reverse direction from u_i to u_{i-1}, i.e. $f(u_i, u_{i-1}) = 1$.*

Remarks: Figure 7 shows a particular example for this lemma.

***Proof*:** First, the proof of sufficiency is obvious. We only prove the necessity below.

Since G^F is a unit-capacity graph and f is an integral flow, for each edge (u,v) in G^F with $c(u,v) = 1$, either $f(u,v) = 0$ or 1. By the definition of $Res(G^F)$, $c_f(u,v) = c(u,v) - f(u,v)$ $\forall (u,v) \in \mathrm{Re}s_f(G^{flow})$ with $f(u,v) = 0$ or -1. Figure 8 enumerates all possible scenarios for converting edges in G^F to the residual graph $Res_f(G^F)$ for flow f, where only the capacity in the residual graph is indicated. The two cases for the edge $e_i = (u_{i-1}, u_i)$ specified in the lemma hold: case 1 corresponds to Figure 8 (a), and case 2 corresponds to Figure 8 (b). □

As stated in Lemmas 1 and 2, a set of node-disjoint paths T can derive a flow f in G^F and the flow f in G^F can also be mapped back to the set T in G. Furthermore, an augmenting path in $Res_f(G^F)$ for the flow f plus the flow along the path can also be mapped back to a path in G with respect to the path set T by merging the companion node pairs in the path and removing any duplicate link between the corresponding root vertices after the conversion. We refer to this path as the *auxiliary path* for the node-disjoint paths T in G. We also say the auxiliary path is mapped from the augmenting path. Notice that edges between non-companion node pairs on the augmenting path in $Res_f(G^F)$ are all retained by the auxiliary path in G. We now give an in-depth analysis in two consecutive steps for this mapping in order to precisely characterize an auxiliary path.

Our first step is to identify the edge changes for each companion node pair from G^F to $Res_f(G^F)$, as shown in Figure 9. Figure 9 (a) demonstrates the conversion when the pair is not on the flow path and Figure 9 (b) demonstrates the conversion when the pair is on the flow path. Notice that when the pair is on the flow path, the following edges are reversed in $Res_f(G^F)$: the egress node connects to the ingress node of the pair, the egress node of the pair is connected from an ingress node of another companion pair also on the flow path, and the egress node of the pair is connected to an ingress node of another companion pair also on the flow

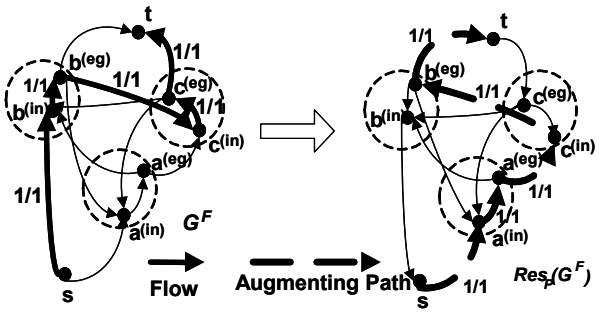

Figure 7 A flow and its augmenting path

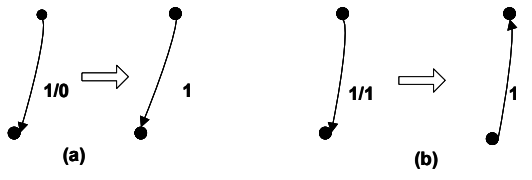

Figure 8 Conversion of edges from G^F to $Res_f(G^F)$ for a flow

path.

Our second step is to study exhaustively the 1:1 mapping between the companion node pair with at least one of the node pair on the augmenting path in $Res_f(G^F)$ and their root node on the auxiliary path in *G,* as shown in Figure 10.

Figure 10 (a) shows that when neither of the companion nodes is on the set of reference paths *T*, the pair must be on the augmenting path. They are directly mapped to their root vertex and the edges attached to the node pair are also mapped directly to the corresponding edges on G, including the two edges on the augmenting path ending and starting at the root node.

Figure 10 (b), (c), (d), and (e) shows that when at least one of the companion node pair is on the set of reference paths *T,* the conversion between the pair and their root node can be elaborated by the following four cases:

1) The egress node is on the augmenting path, but the ingress node is not on the augmenting path. As in Figure 10 (b), the egress node is connected by its successor on the original flow path, which is an ingress node, and it is connected to a different ingress node, which is not its predecessor on the original flow path. As a result, the root node is on the auxiliary path only once; it is first connected from its successor on *T* and then connects itself to a node that is neither its predecessor nor successor on T.
2) The ingress node is on the augmenting path but the egress node is not the augmenting path. As in Figure 10 (c), the ingress node is connected by an egress node that is neither its successor nor its predecessor on the original flow path and is connecting to its predecessor on the original flow path, which is a different egress node. Thus the root node is on the auxiliary path only once; it is first connected from a node that is neither its successor nor predecessor on *T* and then connects itself to its predecessor on *T*.
3) The augmenting path passes through the inner edge connecting the egress node to ingress node (the reverse of the connection in G^F). As in Figure 10 (d), the egress node is connected from its successor on the original flow path that is also an ingress node, and the ingress node is connecting to its predecessor on the original flow path that is also an egress node. Hence the root node is connected from its successor and to its predecessor on *T*.
4) The egress node is on the augmenting path, the ingress node is also on the augmenting path, and the egress node is not directly connected to the ingress node along the augmenting path as shown in Figure 10 (e). Both case 1 and case 2 apply here.

Notice that the two cases 1 and 3 can be combined: if a node is on both the auxiliary path and the path set *T*, and its predecessor on the auxiliary path is its successor on *T*, then the successor of the node on the auxiliary path can be any node other than itself.

Summarizing the above discussion, we can obtain the following conclusion:

Lemma 4 *An auxiliary path in G for an existing set of node-disjoint paths T consists of the following two types of edges:*

1) *Both the edge (u, v) and its reverse edge (v, u) are not edges in the set of node-disjoint paths T.*
2) *Only the reverse edge (v, u) is in the set of node-disjoint paths; the auxiliary path edge (u, v) is not.*

The above two types of edges must also satisfy the following requirement:

3) *For a vertex u on both the auxiliary path and on the reference path T, let p→u→v be a segment on the auxiliary path. Two cases can occur:*
   - ✓ *u is on the auxiliary path twice: In one appearance, p must be a successor of u on T and v must be neither the successor nor the predecessor of u on T, i.e. (u, p) is an edge in the path set T but both (u, v) and (v, u) are not on T. In another appearance, p must be neither a successor nor a predecessor of u on T and v must be the predecessor of u on T, i.e. both edges (p, u) and (u, p) are not on T but (v, u) is on T.*
   - ✓ *u is on the auxiliary path only once: if p is not the successor (as well as not its predecessor ob-*

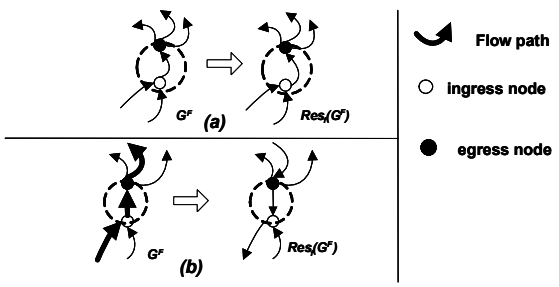

Figure 9 Companion node pair, and their associated edges

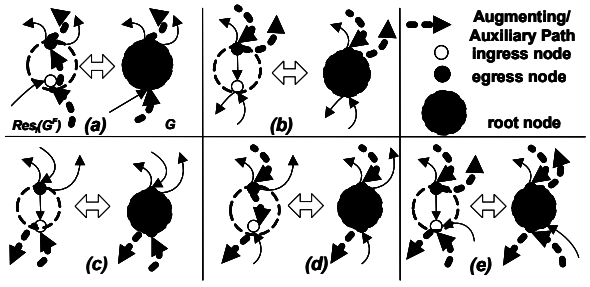

Figure 10 Complete vertex conversion table between $Res_f(G^F)$ and G

viously) of u on T, then v must be the predecessor of u on T, i.e. if both edges (p, u) and (u, p) are not on the path set T, the edge (v, u) must be on T.

In the next section, we will utilize the mapping, as described in Figure 10 and discussed above, to convert the procedure for discovering an augmenting path in $Res_f(G^F)$ to a corresponding procedure in G for discovery of an auxiliary path. Figure 11 shows a particular example for mapping an augmenting path in $Res_f(G^F)$ (Figure 11 (a) and (b)) to an auxiliary path in *G* (Figure 11 (c)).

D. Merging Auxiliary Path into Node-disjoint Paths

There is 1:1 mapping between auxiliary paths in *G* for a set of node-disjoint paths *T* and augmenting paths in $Res_f(G^F)$ for the paths in *T* derived flow *f*. In other words, identifying an augmenting path in $Res_f(G^F)$ is equivalent to discovering such an auxiliary path in *G* for the path set *T*. However, a new auxiliary path is not necessarily node-disjoint with all paths in *T*. Constructing new node-disjoint paths in *G* from an auxiliary path requires merging, or even reorganizing, the auxiliary path and paths in *T*. By Lemma 4 and a mapping from the flow merging rules in G^F for the set of node-disjoint paths *T*, we derive the following procedure to obtain new node-disjoint paths from an augmenting path *P*.

Merging Algorithm for Auxiliary Path

1. *Create a directed graph $H = (V_H, E_H)$ so that it only contains the existing node-disjoint paths from the s to t.*
2. *Add the vertices and edges of the auxiliary path in G to H. Denote the resulting graph as $H = (V_H, E_H)$.*
3. *For each edge (u, v) in E_H, if (v, u) is also in E_H, remove the two edges (u, v) and (v, u) from E_H. Again denote the resulting graph as $H = (V_H, E_H)$.*

After Step 3, the graph *H* will only consist of node-disjoint paths from *s* to *t*. *H* will include one more node-disjoint path than *G*. Figure 12 gives an example of the above procedure.

Based on the Max-Flow Min-Cut theorem [1] and the results of Lemmas 1 through 4, we obtain our primary result in the following theorem and corollary.

Theorem 1 *Let T be a set of node-disjoint paths in the ad hoc network G=(V, E) between source s and destination t. Let P be an auxiliary path for T in G. Merge T and P by the Merging Algorithm for the auxiliary path and get a new path set T' between s and t. The new path set T' is node-disjoint between s and t in G and* $|T'| = |T| + 1$.

Corollary 1 *Let T be a set of node-disjoint paths in the ad hoc network G=(V, E) between source s and destination t. Then T has the maximal number of node-disjoint paths between s and t if and only if there are no more auxiliary paths.*

From Theorem 1 and Corollary 1**,** we conclude that for a set of existing node-disjoint paths between a source and a destination, if we want to find additional node-disjoint paths, we must find an auxiliary path first and then merge the path with the existing path set to get the new set of node-disjoint paths. If an auxiliary path does not exist, we know deterministically that the maximal number of node-disjoint paths has already been found.

E. On-Demand Discovery of Node-Disjoint Paths

Based on Theorem 1, we can easily derive a general, iterative, and hence on-demand algorithm for finding multiple node-disjoint paths incrementally in an ad hoc network:

MNDP Algorithm

G: An ad hoc network with source *s* and destination *t.*
T: The current set of node-disjoint paths in *G* between *s* and *t*.

1. $T = \phi$;
2. *Find an auxiliary path P in G from s to t for the path set T;*

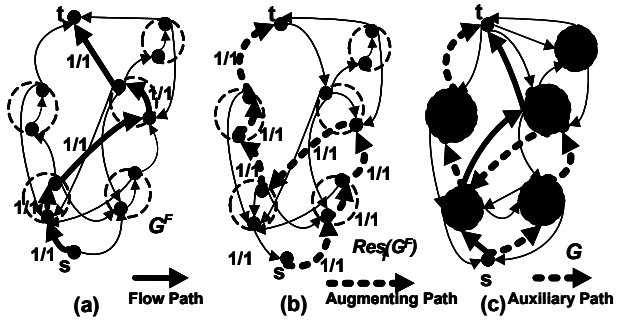

Figure 11 (a) A flow in G^F. (b) An augmenting path for the flow. (c) Mapping of the augmenting path to an auxiliary path in G.

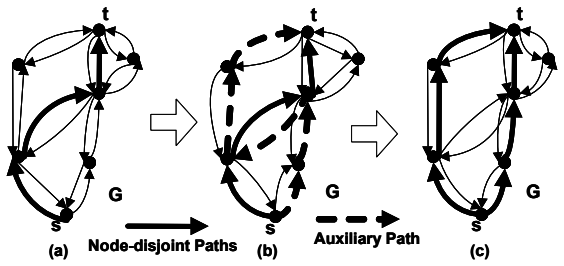

Figure 12 (a) Graph *G* with a path from *s* to *t*. (b) Find an auxiliary path in G. (c) Get 2 node-disjoint paths after running the merging algorithm.

3. *Merge P with T by the Merging Algorithm to get a new path set in G, still denoted as T.*
4. *Repeat steps 2 and 3 until either finding the desired number of node-disjoint paths or until no additional auxiliary paths can be found in G.*

Notice that the MNDP algorithm is an iterative process. It can start from an empty path set or an existing path set, and it stops when a targeted number of node-disjoint paths have been discovered. The algorithm does not assume global knowledge of the ad hoc network, nor does it require periodic information exchange. To increase the number of node-disjoint paths by one, a new auxiliary path can be discovered on-demand based on existing paths, and a new set of node-disjoint paths can be obtained after the merging procedure.

IV. Example On-Demand Two-Node-Disjoint Path Routing Protocol

In this section, we present an example on-demand routing protocol that discovers two node-disjoint paths using the theoretical framework in Section III. The protocol, called the Auxiliary Path Discovery Protocol (APDP) is a derivative of DSR [4] integrated with the MNDP algorithm for the discovery and maintenance of multiple paths. While this example focuses on discovering two node-disjoint paths, it can be easily extended to support *k* node-disjoint paths.

A. APDP Route Discovery

The procedure used by APDP for discovering two node-disjoint paths from source *s* to destination *t* is iterative and has three phases. In Phase 1, an initial path (referred to as the reference path) from node *s* to node *t* is discovered using the basic approach from DSR [4]. In Phase 2, an auxiliary path from *s* to *t* is discovered based on the reference path. The auxiliary path will satisfy Lemma 4. Finally, in Phase 3, the Merging Algorithm (Section III.D) is applied to merge the reference and auxiliary path, resulting in two node-disjoint paths from *s* to *t* in the network.

Following the discovery of a reference path by using unmodified DSR, auxiliary path discovery requires comparison and processing of potential edges in the auxiliary path to edges in the reference path (see Lemma 4). To enable such comparison, we modify the DSR RREQ message by adding a list of nodes representing the reference path from *s* to *t*. This modified message format is referred to as RREQ_A. The following auxiliary path discovery procedure between *s* and *t* is derived from the mapping between the augmenting path and the auxiliary path analyzed earlier (Figure 10 and Lemma 4, in Section III.C). In particular, the problem of finding augmented paths in $Res_f(G^F)$ can be mapped into use of DSR flooding over *G* to find an auxiliary path using the following procedure.

To initiate discovery of an auxiliary path, the source node *s* creates a RREQ_A message as in basic DSR, except that the reference path is filled with the results of the reference path discovery from Phase 1. Let *u* be the current node and *p* be the previous node in the path that forwarded the RREQ_A message to *u*. When a node receives a RREQ_A message, it is processed according to the following four rules, depending on whether node *u* is listed in the reference path.

Condition 1: If node *u* is listed in the reference path in the RREQ_A, the node checks if the previous node *p* is its successor or predecessor on the reference path. If *p* is neither its successor nor its predecessor on this reference path, the node caches the RREQ_A, appends its address to the auxiliary path (i.e. route record), and unicasts the RREQ_A to its predecessor on the reference path. This roughly corresponds to Figure 10 (c) and (e).

Condition 2: If *p* is *u*'s successor on the reference path, the node caches the RREQ_A, appends its address to the auxiliary path, and broadcasts the message to its neighbors. Otherwise, the RREQ_A is dropped since this condition fails Lemma 4, rule 2. This roughly corresponds to Figure 10 (b), (d), and (e).

Condition 3: If node *u* is not listed in the reference path, it caches the RREQ_A, appends its address to the auxiliary route record, and broadcast the message to its neighbors. This roughly corresponds to Figure 10 (a).

Condition 4: If the node has a record in its cache of the same RREQ_A that was previously received under the same condition, it drops the RREQ_A.

When the destination node *t* receives the RREQ_A, it responds with a RREP_A, which is identical to a DSR RREP message with the addition of the reference path. The RREP_A can be sent back either along the reference path or along the auxiliary path in the route record. When the source node *s* receives the RREP_A, it executes Phase 3 of route discovery by merging the reference path with the auxiliary path included in the RREP_A using the Merging Algorithm. Using this process, the APDP protocol will discover two node-disjoint paths if they exist in the network. As an alternative, the destination node could instead execute the Merging Algorithm to obtain two node-disjoint paths from *t* to *s*.

While the algorithm described above discovers exactly two node-disjoint paths (when present), it is easy to extend this algorithm to find k node-disjoint paths. For each additional node-disjoint path that is desired, Phases 2 and 3 (above) can be repeated. The RREQ_A (and RREP_A) packet must be further augmented to contain the i reference paths that will be known during the i^{th} iteration of Phases 2 and 3. The conditions for handling RREQ_A must be applied to all reference paths present in the RREQ_A packet. By simply executing iterations until Phase 2 fails to find a new auxiliary path, the same algorithm can be used to identify all node-disjoint paths between nodes t and s.

B. Route Maintenance

APDP supports two-route maintenance with a different procedure depending on whether only one node-disjoint path fails or both node-disjoint paths fail.

When only one of the two node-disjoint paths from s to t is broken, node s repairs the broken path by re-executing Phases 2 and 3 using the other valid path as the reference path field in the RREQ_A. In this case, the path that remains after the failure can be used in place of the successful result of a Phase 1. After Phases 2 and 3, two node-disjoint paths from s to t will be discovered (if present).

When both node-disjoint paths fail simultaneously, node s restarts Phases 1-3 to find two new node-disjoint paths to t.

C. Additional Route Discovery Features

DSR optimization features such as promiscuous route caching, Route Reply storm prevention, and limited propagation of Route Requests can be applied in the first two phases of APDP without modification. However, allowing intermediate nodes to reply to Route Requests using cached routes requires modification for correct operation in Phase 2 of APDP. These changes are detailed below.

As is the case in DSR, a node receiving a RREQ_A for which it is not the destination may search its own Route Cache for a route to the destination of the RREQ_A. In the second phase of the APDP, if a cached route is found, the node concatenates the route record with the source route and checks for the following two conditions:

- *For each edge e=(u,v) in the concatenated source route to the target, verify that the edge e is not in the reference path node list from s to t.*
- *Verify that Rule 3 in Lemma 4 satisfied.*

If both of the above conditions are true, the intermediate node can return a RREP_A to the source of the RREQ_A rather than forwarding the RREQ_A. In the RREP_A, the node sets the auxiliary route record to list the sequence of nodes over which this copy of the RREQ_A was forwarded, concatenated with the cached source route to the destination.

D. Example APDP path discovery

In Section 2, we showed that existing ad hoc networking multipath protocols were not able to guarantee discovery of two node-disjoint paths in the example topologies from Figure 2 and Figure 3. We now demonstrate that APDP is able to discover two node-disjoint paths in both examples.

Figure 13 shows how the APDP discovers two node-disjoint paths between nodes 23 and 5 for the example topology from Figure 2. For simplicity, only the relevant network sub-topology is drawn in this figure. In Phase 1, the reference path is discovered using unmodified DSR route discovery, resulting in selection of one of the shortest path routes, e.g., 23→21→7→0→5 or 23→21→11→1→5. For the sake of discussion, assume the first route is selected, as in Figure 13 (a). In Phase 2, APDP discovers an auxiliary path for the reference path 23→21→7→0→5. For example, 23→3→26→7→21→11→0→7→1→5 and 23→3→26→30→11→1→5 are two auxiliary paths that could be discovered by the protocol, depending on the RREQ_A propagation pattern. Again assume the first case as in Figure 13 (b). Finally, in Phase 3, the source node 23 merges the reference path (23→21→7→0→5) and the auxiliary path (23→3→26→7→21→11→0→7→1→5) using the Merging Algorithm. The two edges 21→7 (reference path) and 7→21 (auxiliary path) as well as 7→0 (reference path) and 0→7 (auxiliary path) cancel each other in the merging process and the two node-disjoint paths 23→21→11→0→5 and 23→3→26→7→1→5 are discovered as in Figure 13 (c). It is easy to verify that two node-disjoint paths will also be discovered if the second of two shortest paths were identified in Phase 1.

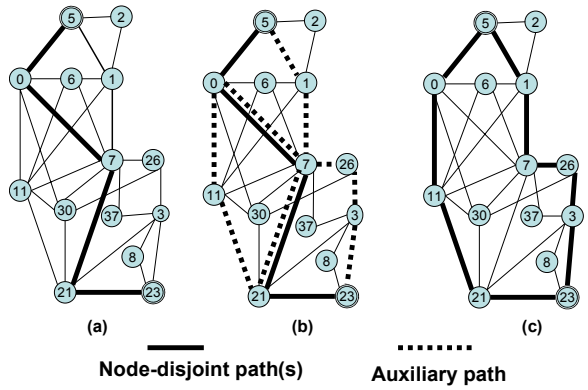

Figure 13 (a) Find one path by DSR. (b) Find an auxiliary path of the path. (c) Merge auxiliary path and the original path.

As a second example, Figure 14 demonstrates how APDP succeeds in discovering two node-disjoint paths for the ad hoc network topology from Figure 3. In Phase 1, APDP discovers the shortest path $s \rightarrow a \rightarrow b \rightarrow c \rightarrow d \rightarrow t$ between s and t as in Figure 14 (a). Next, in Phase 2, it discovers the auxiliary path $s \rightarrow e \rightarrow f \rightarrow g \rightarrow c \rightarrow b \rightarrow h \rightarrow i \rightarrow j \rightarrow t$ as in Figure 14 (b). Finally, in Phase 3, APDP merges the two paths to derive the two node-disjoint paths $s \rightarrow a \rightarrow b \rightarrow h \rightarrow i \rightarrow j \rightarrow t$ and $s \rightarrow e \rightarrow f \rightarrow g \rightarrow c \rightarrow d \rightarrow t$ between *s* and *t* as in Figure 14 (c).

V. Complexity and Performance Measurement

Communication overhead is a good indicator of both complexity and performance for an ad hoc network routing protocol. For on-demand routing protocols, there are two components of communication overhead: unicast messages and broadcast messages.

For the protocols in [4] [5] [14] [15], the unicast overhead is the RREP along the discovered path(s) from destination to source. APDP uses unicast messages for the RREP along the discovered path(s) from destination to source during both Phase 1 and Phase 2, plus at most one message per node on the reference path during RREQ forwarding in Phase 2. In all of these protocols, the unicast overhead is proportional to the broadcast overhead and usually a small fraction of broadcast overhead. Hence, we consider only broadcast messages to compare the communication overhead of these protocols.

The broadcast overhead for APDP, as compared with competing protocols, is shown in Table 1. We evaluate the APDP routing discovery protocol presented in IV, the DSR protocol and other multipath routing protocols [5] [14] against this communication overhead, and argue that our protocol has the best tradeoff in terms of complexity/performance and capability.

Consider an ad hoc network $G=(V, E)$. We show in Figure 15 two near worst case topologies that result in minimum upper bounds for communication overhead for [5] and [14]. Figure 15 (a) shows a particular ad hoc network where the vertex $V - \{t, a, b, c, d, e\}$ forms a complete subgraph. In this figure, the number of broadcasts required by the protocol in [14] for the vertex set $V - \{s, t, a, b, c, d, e\}$ is at least $(|V|-7)(|V|-7) + 1$, which is $O(|V|^2)$. In Figure 15 (b), the vertex set $V - \{s, t, a, b, c, d, e\}$ forms a complete bipartite graph $K_{(|V|-7)/2,(|V|-7)/2}$ with *s* connecting to each vertex of one partition and vertex *a* and vertex *c* connecting to two vertices of the other partition, as shown in the figure. With the vertex set $V - \{s, t, a, b, c, d, e\}$ SMR requires $1 + (|V|-7)/2 + (|V|-7)^2/2$, which is also $O(|V|^2)$. Two node-disjoint paths cannot be discovered by the protocols in [14] and SMR [5] for Figure 15 (a) and (b) since the first discovered path contains the cut {a, c} of the network. Our routing protocol guarantees discovery of two node-disjoint paths for Figure 15 (a) and (b), and it only requires at most $2(|V| - 1)$ broadcasts (the cost of two RREQ floods).

In summary, DSR and AODVM (that employ only one RREQ flood) need $|V| - 1$ broadcasts. This is half of the upper bound for APDP. However, they are not guaranteed to find two node-disjoint paths for an ad hoc network when two such paths exist. SMR [5] and the protocol in [14] also cannot always guarantee discovery of node disjoint paths (as discussed in II), and they require many more broadcasts than APDP in the worst case. Hence, APDP has a good tradeoff in terms of protocol complexity and capability.

VI. Conclusion

This paper presents a theoretical framework for finding *k* node-disjoint paths between a pair of nodes in an on-demand manner. We transform the problem of finding *k* node-disjoint paths to the problem of finding a flow assignment of value *k* in a flow network. We propose procedures to incorporate the Ford-Fulkerson method into on-demand discovery of *k* node-disjoint paths in an ad hoc network. Furthermore, we present an example protocol that integrates DSR with the theoretical framework to find two node-disjoint paths. Analysis of the protocol overhead demonstrates a good tradeoff between protocol complexity and capability, par-

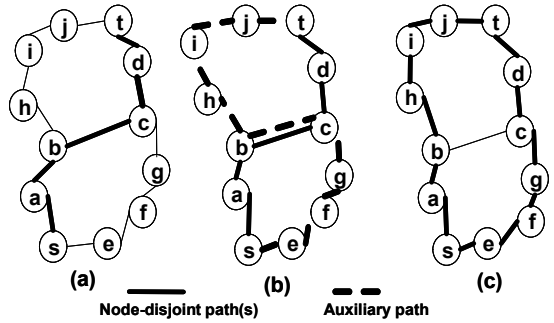

Figure 14 (a) Find one path by DSR. (b) Find an auxiliary path of the path. (c) Merge auxiliary path and the original path.

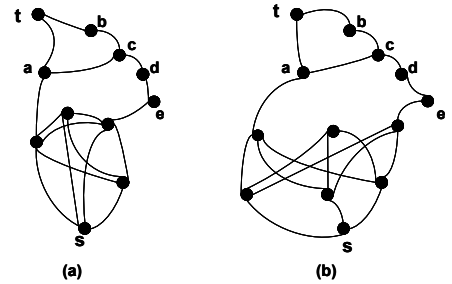

Figure 15: Examples of ad hoc networks.

TABLE 1: COMMUNICATION OVERHEAD OF MULTIPATH ROUTING PROTOCOLS

	RREQ Forwarding Overhead	Number of Broadcasts
DSR [4]	Each node except the destination broadcasts RREQ one time.	$\|V\|$-1
APDP	Each node except the destination broadcasts RREQ one time at phase 1 and at most 1 broadcast at phase 2.	at most 2($\|V\|$-1)
SMR [5]	Each node v may forward at most $deg(v)$ broadcasts, where $deg(v)$ is the number of the neighboring nodes of node v.	at least $\|V\| - 1$, at most $\max_{v \in V-\{s,t\}} deg(v) * (\|V\| - 2) + 1 - deg(t)$
Routing Protocol in [14]	Each node v may forward at most $deg(v)$ broadcasts.	at least $\|V\| - 1$, at most $\max_{v \in V-\{s,t\}} deg(v) * (\|V\| - 2) + 1 - deg(t)$
AODVM [15]	Each node except the destination broadcasts RREQ one time.	$\|V\|$-1

ticularly as compared with other on-demand multipath routing protocols. To the best of our knowledge, our framework is the first to guarantee the on-demand discovery of node-disjoint paths in wireless ad hoc networks.

REFERENCES

[1] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*, The MIT Press, 1993.

[2] D. S. J. De Couto, D. Aguayo, J. Bicket, and R. Morris, "A High-Throughput Path Metric for Multi-Hop Wireless Routing," *Proceedings of the 9th ACM International Conference on Mobile Computing and Networking (MobiCom '03)*, San Diego, CA, September 2003.

[3] D. Ganesan, R. Govindan, S. Shenker, and D. Estrin, "Highly Resilient, Energy Efficient Multipath Routing in Wireless Sensor Networks", *Mobile Computing and Communications Review*, Vol. 1, No. 2 (2002).

[4] D. B. Johnson, D. A. Maltz, Y. Hu, and J. Jetcheva. "The Dynamic Source Routing protocol for mobile ad hoc networks." IETF Internet Draft, draft-ietf-manet-dsr-09.txt, Oct 2003.

[5] S.J. Lee and M. Gerla "SMR: Split Multipath Routing with Maximally Disjoint Paths in Ad hoc Networks", *Proceedings of ICC 2001, Helsinki, Finland, June 2001.*

[6] M. K. Marina and S. R. Das, "On-demand Multipath Distance Vector Routing in Ad Hoc Networks", *Proceedings of the International Conference for Network Protocols (ICNP 2001)*, Riverside, CA, November 2001.

[7] MIT Roofnet Project, http://www.pdos.lcs.mit.edu/roofnet/, June 12, 2004.

[8] A. Nasipuri, R. Castaneda, and S. R. Das, "Performance of Multipath Routing for On-Demand protocols in Mobile Ad Hoc Networks," *ACM/Baltzer Mobile Networks and Applications (MONET) Journal*, vol. 6, pp. 339-349, 2001.

[9] A. Nasipuri and S. R. Das, "On-Demand Multipath Routing for Mobile Ad Hoc Networks", *Proceedings of the 8th Int. Conf. On Computer Communications and Networks (IC3N)*, Boston, MA, October 1999.

[10] C. E. Perkins and E. M. Royer, "Ad hoc On-Demand Distance Vector Routing," *Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications*, New Orleans, LA, February 1999.

[11] P. P. Pham and S. Perreau, "Performance Analysis of Reactive Shortest Path and Multi-path Routing Mechanism With Load Balance", *Proceedings of IEEE INFOCOM 2003*, San Francisco, CA, March 2003.

[12] Seattle Wireless Community Network Project, http://www.seattlewireless.net/.

[13] S. Vutukury and J. J. Garcia-Luna-Aceves, ``Mdva: A distance-vector multipath routing protocol," *Proceedings of IEEE INFOCOM 2001*, Anchorage, AK, April 2001.

[14] K. Wu and J. Harms, "On-Demand Multipath Routing for Mobile Ad Hoc Networks," *Proceedings of 4th European Personal Mobile Communication Conference (EPMCC 01)*, Vienna, Austria, Feb. 2001.

[15] Z. Ye, S. V. Krishnamurthy, and S. K. Tripathi, "A Framework for Reliable Routing in Mobile Ad Hoc Networks," *Proceedings of IEEE INFOCOM 2003*, San Francisco, CA, March 2003.

[16] W. Zaumen and J.J. Garcia-Luna-Aceves, ``Shortest multipath routing using generalized diffusing computations," *Proceedings of IEEE INFOCOM 1998*, San Francisco, CA, March 1998.